

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Щербак Маргарита Романовна

2022

RUDN

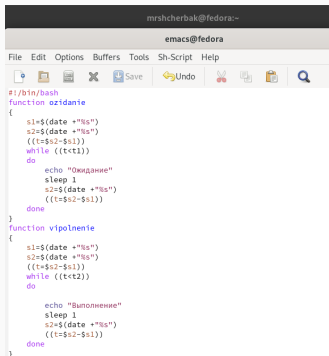
Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Задание:

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

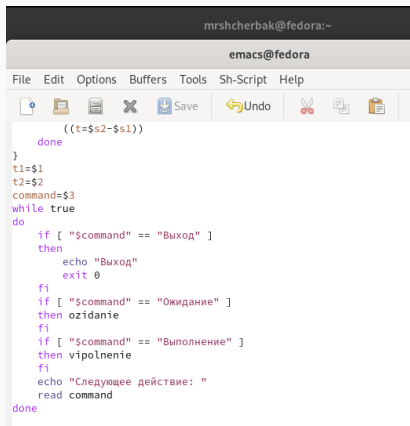
Ход работы: Написала командный файл, реализующий упрощённый механизм семафоров. (Рис. 1 - Рис. 3).

Я создала файл prog11.sh, в котором писала скрипт, открыла текстовый редактор emacs. Также дала созданному файлу право доступа на выполнение (+x).



```
mrshcherbak@fedora:~  
emacs@fedora  
File Edit Options Buffers Tools Sh-Script Help  
#!/bin/bash  
function ozidanie  
{  
    s1=$(date +%s)  
    s2=$(date +%s)  
    ((t=s2-s1))  
    while ((t<t1))  
    do  
        echo "Ожидание"  
        sleep 1  
        s2=$(date +%s)  
        ((t=s2-s1))  
    done  
}  
function vipolnenie  
{  
    s1=$(date +%s)  
    s2=$(date +%s)  
    ((t=s2-s1))  
    while ((t<t2))  
    do  
        echo "Выполнение"  
        sleep 1  
        s2=$(date +%s)  
        ((t=s2-s1))  
    done  
}
```

Figure 1: Скрипт 1го задания



```
((t=$s2-$s1))
done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then ozidanie
    fi
    if [ "$command" == "Выполнение" ]
    then vipolnenie
    fi
    echo "Следующее действие: "
    read command
done
```

Figure 2: Скрипт 1го задания (продолжение)

Делаем вывод, что скрипт работает корректно.

```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ ./prog11.sh 7 3 Ожидание > /dev/tty2 &  
[13] 8330  
  
[12]+  Остановлен      ./prog11.sh 7 3 Ожидание > /dev/tty2  
[mrshcherbak@fedora ~]$ ./prog11.sh 7 3 Ожидание > /home/mrshcherbak/  
bash: /home/mrshcherbak/: Это каталог  
[mrshcherbak@fedora ~]$ ./prog11.sh 7 3 Ожидание /home/mrshcherbak/  
Ожидание  
Ожидание  
Ожидание  
Ожидание  
Ожидание  
Ожидание  
Следующее действие:  
Выполнение  
Выполнение  
Выполнение  
Выполнение  
Следующее действие:  
Выход  
Выход  
  
[13]+  Остановлен      ./prog11.sh 7 3 Ожидание > /dev/tty2  
[mrshcherbak@fedora ~]$ ./prog11.sh 2 6 Ожидание > /dev/tty2 &  
[14] 8422  
[mrshcherbak@fedora ~]$ ./prog11.sh 2 6 Выполнение > /dev/tty2 &  
[15] 8443  
  
[14]+  Остановлен      ./prog11.sh 2 6 Ожидание > /dev/tty2  
[mrshcherbak@fedora ~]$
```

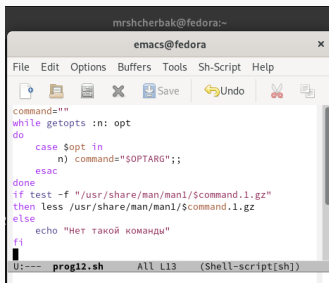
Figure 3: Выполнение

2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.

Командный файл получает в виде аргумента командной строки название команды и в виде результата выдаёт справку об этой команде/сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1. (Рис. 4 - Рис. 8).

Создала файл prog12.sh, в котором писала второй скрипт, и открыла его в редакторе emacs.

Предоставила право доступа на выполнение файлу prog12.sh.



```
command=""
while getopts :n: opt
do
    case $opt in
        n) command="$OPTARG";;
        esac
    done
if test -f "/usr/share/man/man1/$command.1.gz"
then less /usr/share/man/man1/$command.1.gz
else
    echo "Нет такой команды"
fi
```

Figure 4: Скрипт 2го задания

Изучила содержимое каталога /usr/share/man/man1.

```
mrshcherbak@fedora: /usr/share/man/man1
[mrshcherbak@fedora ~]$ cd /usr/share/man/man1
[mrshcherbak@fedora man1]$ ls
.:
.:1.gz
'[:1.gz'
a2ping.1.gz
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
mkfifo.1.gz
mkfontdir.1.gz
mkfontscale.1.gz
mkhybrid.1.gz
mkindex.1.gz
mkisofs.1.gz
mkjobtexmf.1.gz
mkmanifest.1.gz
mkmapfile.1.gz
mknod.1.gz
mkocp.1.gz
mkofm.1.gz
mkpasswd.1.gz
mksquashfs.1.gz
mktemp.1.gz
mktexfmt.1.gz
mktexlsr.1.gz
mktexmf.1.gz
mktexpk.1.gz
mktextfm.1.gz
mlabel.1.gz
mm2gv.1.gz
mmafm.1.gz
mmcli.1.gz
mmd.1.gz
mmdblookup.1.gz
mmount.1.gz
mmove.1.gz
```

Figure 5: Содержимое каталога

Проверка работы скрипта

```
[mrshcherbak@fedora man1]$ cd ~
[mrshcherbak@fedora ~]$ touch prog12.sh
[mrshcherbak@fedora ~]$ chmod +x prog12.sh
[mrshcherbak@fedora ~]$ emacs &
[16] 8681
[mrshcherbak@fedora ~]$ ./prog12.sh -n touch
[mrshcherbak@fedora ~]$ ./prog12.sh -n kill
[mrshcherbak@fedora ~]$ ./prog12.sh -n du
[mrshcherbak@fedora ~]$ ./prog12.sh -n rm
\[mrshcherbak@fedora ~]$ ./prog12.sh -n shshgs
Нет такой команды
[mrshcherbak@fedora ~]$
```

Figure 6: Выполнение

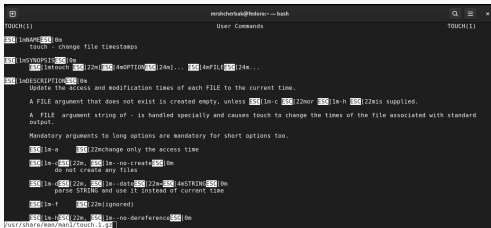


Figure 7: Спрвка по команде touch

Таким образом, мы видим, что задание выполнено успешно.



```
mrshcherbak@fedora:~$ bash
KILL(1)                                User Commands                                KILL(1)

ESC[1mNAMEESC[0m
kill - terminate a process

ESC[1mSYNOPSISESC[0m
ESC[1mkill ESC[22m[-signalESC[1m-s ESC[4mESC[22msignalESC[24m ESC[1m-pESC[22m] ESC[1m-q ESC[4mESC[22mvalueESC[24m] ESC[1m-a
ESC[22m] ESC[1m--timeout ESC[4mESC[22millisecondsESC[24m ESC[4msignalESC[24m] ESC[1m--ESC[22m ESC[4mpidESC[24m] ESC[4mnameESC[24m...

ESC[1mkill -l ESC[22mESC[4mnumberESC[24m] | ESC[1m-LESC[0m

ESC[1mDESCRIPTIONESC[0m
The command ESC[1mkill ESC[22msends the specified ESC[4msignalESC[24m to the specified processes or process groups.

If no signal is specified, the TERM signal is sent. The default action for this signal is to terminate the process. This
signal should be used in preference to the KILL signal (number 9), since a process may install a handler for the TERM
signal in order to perform clean-up steps before terminating in an orderly fashion. If a process does not terminate after a TERM
signal has been sent, then the KILL signal may be used; be aware that the latter signal cannot be caught, and so does not
give the target process the opportunity to perform any clean-up before terminating.

Most modern shells have a builtin ESC[1mkill ESC[22mcommand, with a usage rather similar to that of the command described here.

The
ESC[1m--allESC[22m, ESC[1m--pidESC[22m, and ESC[1m--queueESC[22moptions, and the possibility to specify processes by command na
me, are local extensions.

If ESC[4msignalESC[24m is 0, then no actual signal is sent, but error checking is still performed.

ESC[1mARGUMENTSESC[0m
The list of processes to be signaled can be a mixture of names and PIDs.
/usr/share/man/man1/kill.1.gz
```

Figure 8: Справка по заданной команде

- Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Используя встроенную переменную \$RANDOM, написала командный файл, генерирующий случайную последовательность букв латинского алфавита. (Рис. 9 - Рис. 10).

Создала файл prog13.sh, в котором писала третий скрипт, и открыла его в редакторе emacs. Предоставила право доступа на выполнение файлу prog13.sh.

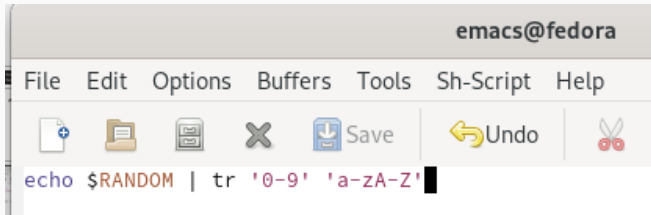


Figure 9: Скрипт 3го задания

Скрипт работает корректно.

```
[mrshcherbak@fedora ~]$ touch prog13.sh
[mrshcherbak@fedora ~]$ chmod +x prog13.sh
[mrshcherbak@fedora ~]$ emacs &
[17] 9040
[mrshcherbak@fedora ~]$ ./prog13.sh
ciahb
[mrshcherbak@fedora ~]$ ./prog13.sh
dabhh
[mrshcherbak@fedora ~]$ ./prog13.sh
chghj
[mrshcherbak@fedora ~]$ ./prog13.sh
bbdcc
[mrshcherbak@fedora ~]$ ./prog13.sh
bida
[mrshcherbak@fedora ~]$ ./prog13.sh
cddih
[mrshcherbak@fedora ~]$ ./prog13.sh
gacd
[mrshcherbak@fedora ~]$ □
```

Figure 10: Выполнение

Таким образом, в ходе ЛРН№12 я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.